AMENDMENTS TO THE CLAIMS:

This listing of claims will replace all prior versions, and listings, of claims in the application:

1. (Previously Presented) A method of enabling multiple different operating systems to run concurrently on the same computer, said method comprising:

selecting a first operating system to have a high priority;

selecting at least one second operating system to have a lower priority ;

providing a common program arranged to switch between said operating systems under predetermined conditions; and

providing modifications to said first and second operating systems to allow them to be controlled by said common program;

wherein switching between said operating systems occurs by responding to real or virtual events and includes invoking the common program by calling an exception vector, and

wherein calling an exception vector to invoke the common program simulates an exception caused by an external event.

Claim 2 (Canceled).

3. (Previously Presented) The method of claim 1, comprising allocating exception vectors to trap calls, thereby to enable invocation of the common program using a trap call mechanism.

Claims 4 - 5 (Canceled).

6. (Previously Presented) The method of claim 1, wherein the common program preempts the first or second operating system by intercepting exception or interrupt vectors.

7. (Original) The method of claim 6, further comprising using a exception handler table containing an array of pointers to intercept exceptions, and activating an exception handler program to preempt the first or second operating system.

8. (Previously Presented) The method of claim 1, wherein the common program is executed in real mode in physical processor address space.

9. (Previously Presented) The method of claim 8, comprising preempting the first or second operating system by the common program, and switching to real mode in physical processor address space when preempting the first or second operating system.

10. (Previously Presented) The method of claim 8, comprising invoking the common program by the first or second operating system, and switching to real mode in physical processor address space when invoking the common program.

11. (Previously Presented) The method of claim 1, comprising enabling hardware interrupts throughout the operation of the second operating system except during the operation of subroutines that save machine state.

12. (Previously Presented) The method of claim 1, in which the first operating system is a real time operating system.

13. (Previously Presented) The method of claim 1, in which the second operating system is a non-real time, general-purpose operating system.

14. (Previously Presented) The method of claim 1, in which the second operating system is Linux, or a version or variant thereof.

15. (Previously Presented) The method of claim 1, in which the common program is arranged to save, and to restore from a saved version, the processor state required to switch between the operating systems.

16. (Previously Presented) The method of claim 1, in which processor exceptions for the second operating system are handled in virtual fashion by the common program.

17. (Previously Presented) The method of claim 1, in which the common program is arranged to intercept some processor exceptions, and to call exception handling routines of the first operating system to service them.

18. (Original) The method of claim 17, in which the processor exceptions for the second operating system are notified as virtual exceptions.

19. (Original) The method of claim 18, in which the common program is arranged to call an exception handling routine of the second operating system corresponding to a said virtual exception which is pending.

20. (Previously Presented) The method of claim 1, further comprising providing each of said operating systems with separate memory spaces in which each can exclusively operate.

21. (Previously Presented) The method of claim 1, further comprising providing each of said operating systems with first input and/or output devices of said computer to which each has exclusive access.

22. (Original) The method of claim 21, in which each operating system accesses said first input and/or output devices using substantially modified native routines.

23. (Previously Presented) The method of claim 1, further comprising providing each of said operating systems with access to second input and/or output devices of said computer to which each has shared access.

24. (Original) The method of claim 23, in which all operating systems access said second input and/or output devices using the routines of the first operating system.

25. (Original) The method of claim 24, in which the common program provides trap call mechanisms, to control the operation of the second operating system, and/or event mechanisms to notify the first operating system of status changes in the second operating system.

26. (Previously Presented) The method of claim 1, further comprising combining said operating systems and common program into a single code product.

27. (Previously Presented) The method of claim 1, further comprising embedding said operating systems and common program onto persistent memory on a computer product.

28. (Previously Presented) A development kit computer program product comprising code for performing the steps of claim 1.

29. (Previously Presented) A computer program product comprising code combined according to claim 26.

30. (Previously Presented) A computer system comprising:

a CPU, memory devices and input/output devices, said CPU being arranged to execute computer code comprising:

a first operating system having a relatively high priority;

a second operating system having a relatively lower priority; and

a common program arranged to run said operating systems concurrently by switching between said operating systems under predetermined conditions,

wherein switching between said operating systems occurs by responding to real or virtual events and includes the first or second operating system invoking the common program by calling an exception vector, wherein calling an exception vector to invoke the common program simulates an exception caused by an external event.

31. (Previously Presented) A computer system according to claim 30, arranged to run said first and second operating systems concurrently by:

selecting a first operating system to have a high priority;

selecting at least one second operating system to have a lower priority ;

providing a common program arranged to switch between said operating systems under predetermined conditions; and

providing modifications to said first and second operating systems to allow them to be controlled by said common program;

wherein switching between said operating systems occurs by responding to real or virtual events and includes invoking the common program by calling an exception vector, and

wherein calling an exception vector to invoke the common program simulates an exception caused by an external event.

32. (Previously Presented) The method of claim 1 in which the computer has a Reduced Instruction Set architecture.

33. (New) The method of claim 1, wherein said events are exceptions or interrupts.

34. (New) The computer system of claim 30, wherein said events are exceptions or interrupts.

35. (New) The computer system of claim 31, wherein said events are exceptions or interrupts.